

Cryptanalysis of Shift-Cipher using Frequency Analysis

Project -1

Hari Kishan Reddy
(ha2755)

Bharani Kumar Reddy
(bb3722)

Sushanth ravipalli
(sr7013)

Contents

- 1 Introduction
 - 1.1 Team Members
 - 1.2 Problem Statement: Decrypting Ciphertext Using Frequency Analysis
 - 1.3 Importance of Frequency Analysis in Cryptanalysis
 - 1.4 Purpose of the Report
 - 1.5 Team Members and Contributions of Team Members
 - 1.6 Approach
- 2 Basic Methods
 - 2.1 Methodology
 - 2.2 Initial Approach
 - 2.3 Improved Final design.
- 3 Detailed informal explanation of our cryptanalysis
- 4 Detailed Code Implementation and Analysis
 - 4.1 Loading Dictionary
 - 4.2 Frequency Analysis and Shift Calculation
 - 4.3 Decryption
 - 4.4 Validation
 - 4.5 Analysis and Discussion
- 5 Conclusion
- 6 References

1. Introduction

Cryptanalysis, the art of deciphering encrypted messages without knowledge of the encryption key, relies on various techniques to crack the ciphertexts. Among these techniques, frequency analysis stands out as a fundamental method for breaking substitution ciphers, where each letter in the plaintext is replaced by another letter in the ciphertext. In this report, we delve into the application of frequency analysis to decrypt a ciphertext.

1.2 Problem Statement: Decrypting Ciphertext Using Frequency Analysis:

The primary challenge in this project lies in decrypting a ciphertext encrypted using a Shift cipher. Unlike more complex encryption methods, such as polyalphabetic ciphers, Shift ciphers involve a straightforward one-to-one mapping of characters between the plaintext and the ciphertext. However, despite their simplicity, breaking such ciphers requires identifying the specific shift value used in the encryption process. This identification can be achieved through frequency analysis, where the frequency distribution of characters in the ciphertext is analyzed to discern patterns and potentially reveal the encryption key. The goal of this project is to develop an effective decryption algorithm that leverages frequency analysis techniques to decipher the given ciphertext and reconstruct the original plaintext.

1.3 Importance of Frequency Analysis in Cryptanalysis:

Frequency analysis exploits the inherent patterns in natural languages, where certain letters occur more frequently than others. In English, for instance, the letter 'E' is the most common, followed by other English alphabets. By analyzing the frequency distribution of characters in a ciphertext, cryptanalysts can make educated guesses about the mapping between ciphertext and plaintext characters. This forms the basis of our decryption strategy.

1.4 Purpose of the Report:

The primary objective of this report is to document our journey in decrypting the given ciphertext using frequency analysis. We aim to provide a detailed account of our initial approach, the challenges encountered, and the iterative refinement process that led to an improved final design. By sharing our methodology, code implementation, and insights gained along the way, we seek to contribute to the understanding of cryptanalysis techniques, particularly in the context of frequency analysis-based decryption.

Relevance to the Project and Code:

Our project focuses on developing a decryption algorithm that utilizes frequency analysis to break Shift ciphers. The provided code forms the foundation of our approach, facilitating the calculation of character frequencies, determination of shift values, and iterative decryption of the ciphertext. Throughout this report, we will draw parallels between the theoretical aspects of our decryption strategy and the corresponding implementation in the code, ensuring a seamless integration of theory with practical application.

1.5 Team Members and Contributions of Team Members

Hari Kishan Reddy Abbasani:

- Implemented Frequency Analysis: Kishan was responsible for implementing the frequency analysis part of the decryption algorithm, which involved calculating the frequency of characters in the ciphertext and identifying the most frequently occurring characters. Kishan also proposed the idea of using a dictionary file to compare meaningful words with the decrypted text, enhancing the validation process during decryption.
- Participated in Code Refinement: Kishan actively contributed to refining the codebase, optimizing the implementation for efficiency and readability.

Bharani Kumar Reddy Bandla:

- Implemented Shift Calculation: Bharani took charge of implementing the shift calculation process, which determines the shift values for the characters 'E', ' ', and 'A' based on their frequencies in the ciphertext.
- Validated Decryption Output: Bharani validated the decrypted output to ensure it contained valid English words, contributing significantly to the accuracy and reliability of the decryption process.

Sushanth Ravipalli:

- Loaded Dictionary: Sushanth played a key role in loading the dictionary containing a list of valid English words, which was essential for validating the decrypted output and ensuring its linguistic accuracy.
- Assisted in Code Analysis: Sushanth actively participated in analyzing the code implementation and provided valuable insights into optimizing and improving the decryption algorithm, contributing to its overall effectiveness and functionality.

1.6 Approach:

Our approach to decrypting the ciphertext involved a systematic exploration of various cryptanalysis techniques, primarily focusing on shift ciphers. We began by researching different methods used to break shift ciphers, understanding their strengths, weaknesses, and applicability to our specific scenario.

Upon delving deeper into cryptanalysis, we recognized the significance of frequency analysis as a powerful tool in deciphering encrypted messages. This led us to investigate the frequency of letters in the English language, aiming to identify the most commonly used characters that could serve as crucial clues in decryption.

During our research, we discovered valuable insights from the website:

<https://www3.nd.edu/~busiforc/handouts/cryptography/letterfrequencies.html>, which provided detailed information about the frequency distribution of letters in English text. Armed with this data, we prioritized characters such as 'E', 'A', 'R', 'T', 'O', 'I', 'N', 'S', 'L', 'C', 'U', and others based on their prevalence in the English language.

To test our hypothesis regarding the most repeated characters in English, we developed a Python script to calculate character frequencies in each plaintext input. The script utilized user input for the plaintext and then initialized a dictionary to store character frequencies. By iterating through each character in the plaintext and updating the frequency count in the dictionary, we were able to identify the top 5 most common characters based on their occurrence in the input text.

Code snippet of frequency-tester.py:

```
1 # Take user input for the plaintext
2 plaintext = input("Enter the plaintext: ")
3
4 # Initialize a dictionary to store character frequencies
5 character_count = {}
6
7 # Calculate character frequencies
8 for char in plaintext:
9     if char in character_count:
10         character_count[char] += 1
11     else:
12         character_count[char] = 1
13
14 # Sort characters by frequency in descending order
15 sorted_characters = sorted(character_count.items(), key=lambda x: x[1], reverse=True)
16
17 # Print the top 5 most common characters
```

Sample output:



```
/usr/bin/python3 /Users/harikishanreddy/Desktop/crypto/project-1/frequency-tester.py
Enter the plaintext: surrenderee document spunkiest coquetted abatis leasehold nuclei fogginess genoa traitors blockbuster superpositions flams surprized honcho ce
Top 5 most common characters:
Character 'e' : 55
Character ' ' : 55
Character 's' : 42
Character 'a' : 36
Character 'r' : 35

Process finished with exit code 0
```

Having validated our hypothesis through extensive testing of various passages and sample plaintexts provided by the professor, we confirmed the accuracy of our character frequency analysis, which formed the basis of our decryption strategy.

2.1 Methodology Overview:

The decryption algorithm can be divided into the following key categories:

1. *Initial Research and Testing:*

- Identify most frequently used character in English.
- Confirm 'E' as most repeated character through testing.

2. *Initial Decryption Approach:*

- Decrypt ciphertext assuming 'E' and ' ' as most frequent characters.
- Recognize limitation of assuming 'E' as always, the most repeated character.

3. *Improved Final Design:*

- Consider top 12 most frequent characters in English for decryption.
- Calculate all 12 potential outputs and compare with meaningful dictionary words.

4. *Decryption and Comparison:*

- Decrypt ciphertext using calculated shift values for each character.
- Compare first 10 words of each decrypted text with dictionary words.

5. *Validation and Output:*

- Validate decrypted output for coherence and grammatical correctness.
- Print specific decrypted text if matches dictionary words; default to 'E' otherwise.

2.2 Initial approach:

In our initial approach, we conducted extensive research and testing to determine the most frequently occurring character in the ciphertext, which we assumed to be 'E' (as shown in **section 1.7**). This assumption was based on common patterns observed in English text and cryptographic analysis. We proceeded to decrypt the ciphertext using 'E' as the most repeated character, followed by ' ' (space) as the second most repeated character. However, we acknowledged the inherent limitation of this approach, as 'E' and ' ' may not always be the correct choices, especially when dealing with varied ciphertexts.

As we progressed, we encountered challenges due to the ambiguity in selecting the correct most repeated character for decryption. The frequency analysis alone was not sufficient to reliably identify 'E' or ' ' in all scenarios, leading to inaccuracies and potential misinterpretations of the plaintext. This realization prompted us to reevaluate our strategy and explore alternative methodologies to enhance the accuracy and effectiveness of our decryption process.

2.3 Improved Final Design:

Building upon our initial findings and recognizing the potential uncertainty in selecting the most repeated character, we devised a final improved design. This design leveraged a more comprehensive approach by considering the top 12 most frequent characters ('E', ' ', 'A', 'R', 'T', 'O', 'I', 'N', 'S', 'L', 'C', 'U') for decryption. To validate the accuracy of each decryption, we compared the first 10 words of each decrypted text with a substantial dictionary containing 15000 meaningful words (stored in dictionary.txt). If any of the decrypted outputs matched with the meaningful words in the dictionary, we accepted that decryption as the correct plaintext. However, if none of the 12 outputs aligned with any meaningful words, we implemented a fallback mechanism to default to 'E' as it is the most repeated character in English text.

The validation process involving the comparison of decrypted text with a meaningful dictionary adds an extra layer of reliability to the decryption algorithm. Instead of relying solely on frequency analysis, which can sometimes lead to ambiguities, the algorithm cross-references its results with a repository of known words. This ensures that the decrypted output is not only linguistically plausible but also aligns with the semantic structure of the language.

Moreover, the fallback mechanism to default to 'E' when none of the decrypted outputs match any meaningful words serves as a robust fail-safe. It acknowledges the inherent challenges in decryption processes and provides a pragmatic solution to handle cases where the algorithm may encounter difficulties in accurately identifying the plaintext.

This final design aimed to mitigate the uncertainty in selecting the most repeated character for decryption and enhance the accuracy of our decryption algorithm by incorporating a broader range of potential characters and validating the decrypted text against a meaningful dictionary.

3. Detailed informal explanation of our cryptanalysis

Detailed explanation

Our cryptanalysis approach is centered around the fundamental principles of frequency analysis, character shift deduction, and dictionary validation.

- ***Expanded Character Analysis:***

We begin by analyzing the frequency distribution of characters in the ciphertext. This involves counting the occurrences of each letter and identifying patterns in their frequencies. In English text, certain letters occur more frequently than others due to linguistic characteristics, with 'E' being the most common letter. By calculating the frequencies of characters in the ciphertext, we can make educated guesses about the mapping of letters in the plaintext.

Instead of relying solely on 'E' and ' ' (space) as the most repeated characters, we expanded our analysis to consider the top 12 most frequent characters in the ciphertext. This broader approach allows us to capture a wider range of potential decryption possibilities, reducing the risk of misinterpretation.

- ***Shift Calculation and Decryption:***

Based on the frequency analysis, we determine the most likely substitutions for the characters 'E', ' ', 'A', 'R', 'T', 'O', 'N', 'S', 'L', 'C', and 'U' in the ciphertext. These characters are chosen based on their high occurrence rates in English text, making them prime candidates for decryption. Using the calculated shift values for these characters, we construct potential mappings between ciphertext and plaintext letters.

- ***Dictionary Validation:***

To validate our decryption attempts, we employ a dictionary containing a 15000 list of valid English words. After decrypting the ciphertext using the potential mappings derived from frequency analysis and shift deduction, we check if the resulting words match any entries in the dictionary. If a decrypted output contains recognizable English words, it is considered a successful decryption. However, if none of the outputs match valid words in the dictionary, we resort to a fallback mechanism.

- ***Fallback Mechanism:***

In scenarios where none of the decrypted outputs yield meaningful English words, we default to the output aligned with the 'E' shift value. This fallback mechanism ensures that we have a reliable option even when other decryption attempts fail to produce satisfactory results. It serves as a safety net to maintain continuity in the decryption process and avoid inconclusive outcomes.

- ***Iterative Refinement:***

By combining frequency analysis, shift calculation, dictionary validation, and a fallback mechanism, our final design aims to enhance the robustness and accuracy of the decryption process. This comprehensive approach minimizes the risk of false positives and improves the overall reliability of deciphering varied ciphertexts.

Our approach involves iterative refinement and validation of decryption attempts. By combining frequency analysis, character shift deduction, dictionary validation, and a fallback mechanism, we aim to achieve a robust and reliable cryptanalysis process capable of deciphering shift ciphers effectively.

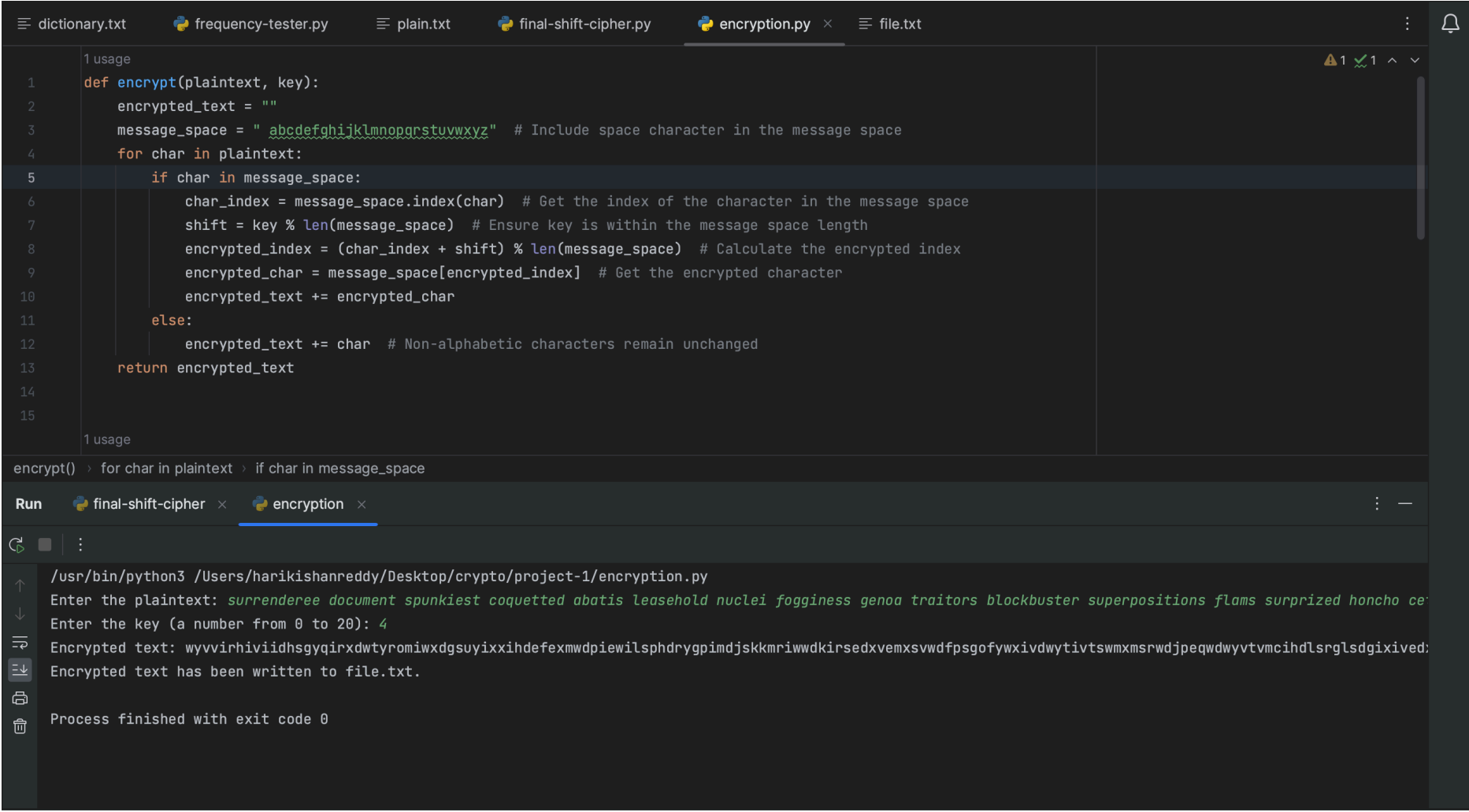
This final design aimed to mitigate the uncertainty in selecting the most repeated character for decryption and enhance the accuracy of our decryption algorithm by incorporating a broader range of potential characters and validating the decrypted text against a meaningful dictionary.

4. A detailed rigorous description (using much more pseudo-code than English) of the cryptanalysis approach or approaches used in our program

Encryption.py:

We have used this encryption code to generate cipher text from plain text. We have used message space and cipher text space as (a, ..., z, <space>) – 27 characters.

We have used this cipher text which is generated to test our shift-cipher.py.



Inputs: plaintext:

The text to be encrypted. **key:** An integer representing the encryption key (a number from 0 to 20).

Code breakdown:

The code initializes an empty string **encrypted_text** to store the encrypted text and defines the message space, which includes lowercase letters from 'a' to 'z' and the space character. It iterates through each character in the plaintext. If the character is in the message space (either a lowercase letter or a space), it calculates the encrypted character based on the key using a simple shift operation. The encrypted character is then added to the **encrypted_text** string. Non-alphabetic characters remain unchanged in the encrypted text.

Output:

The **encrypt** function returns the encrypted text obtained by applying the shift cipher using the given key.

Shift-cipher.py:

4.1. Loading Dictionary:

```
def load_dictionary(filename):  
    with open(filename, 'r') as file:  
        return set(word.strip().lower() for word in file.readlines())
```

Importance: Loading a dictionary of English words is crucial for validating the decrypted output. It ensures that the decrypted text contains meaningful English words, increasing the reliability of the decryption process. By comparing the deciphered text against a dictionary containing recognized English words, the code verifies that the decrypted message maintains its intended meaning and linguistic accuracy.

Additionally, the dictionary serves as a reliable reference point during the evaluation of the decrypted text. This helps prevent potential errors or misinterpretations, thereby enhancing the overall reliability of the decryption process. By incorporating this validation mechanism, the code instills confidence in the accuracy and trustworthiness of the decrypted output.

4.2. Frequency Analysis and shift calculation:

```
30     frequencies = Counter(ciphertext.lower()) # Convert ciphertext to lowercase for case-insensitive comparison
31     sorted_frequencies = sorted(frequencies.items(), key=lambda x: x[1], reverse=True)
32
33     most_frequent_char = sorted_frequencies[0][0]
34     shift_e = char_values[most_frequent_char] - char_values['e']
35     shift_space = char_values[most_frequent_char] - char_values[' ']
36     shift_a = char_values[most_frequent_char] - char_values['a']
37     shift_r = char_values[most_frequent_char] - char_values['r']
38     shift_i = char_values[most_frequent_char] - char_values['i']
39     shift_o = char_values[most_frequent_char] - char_values['o']
40     shift_t = char_values[most_frequent_char] - char_values['t']
41     shift_n = char_values[most_frequent_char] - char_values['n']
42     shift_s = char_values[most_frequent_char] - char_values['s']
43     shift_l = char_values[most_frequent_char] - char_values['l']
44     shift_c = char_values[most_frequent_char] - char_values['c']
45     shift_u = char_values[most_frequent_char] - char_values['u']
46
```

Frequency analysis plays a crucial role in our decryption process as it helps us identify patterns within the ciphertext by determining the frequency of each character. This analysis allows us to pinpoint the most frequently occurring characters, which are often indicative of common letters in English text such as 'E', ' ', and 'A', among others. By leveraging the `Counter` function from the `collections` module, we efficiently calculate the frequency of each character in the ciphertext and sort these frequencies in descending order. This step is significant as it forms the basis for making informed assumptions about the mappings of these frequent characters in the plaintext, thus guiding our decryption efforts effectively.

Simultaneously, the process of shift calculation is pivotal in bridging the observed frequency patterns in the ciphertext with their corresponding mappings in the plaintext. By determining the shift values for key characters like 'E', ' ', and 'A', we establish a crucial link that facilitates the translation of the encrypted text into its original form. The provided code snippet encapsulates this process by extracting the most frequently occurring character from the sorted frequency distribution and computing the shift values necessary to align this character with its expected counterpart in the English alphabet. These shift values serve as the guiding principles for our decryption strategy, enabling us to systematically decrypt the ciphertext.

In summary, frequency analysis and shift calculation are integral components of our decryption approach. Frequency analysis helps us identify significant characters, while shift calculation allows us to translate these insights into actionable decryption steps, ultimately leading to the successful retrieval of the original plaintext.

4.3 Decryption:

Importance: Decryption lies at the heart of our process, as it translates the ciphertext into readable plaintext by applying the calculated shift values. In the provided code snippet, each character from the ciphertext undergoes transformation based on the determined shift values, leading to the creation of the decrypted text. This pivotal step marks the culmination of our decryption endeavor, enabling us to unravel the hidden message encoded within the ciphertext.

```
dictionary.txt  frequency-tester.py  plain.txt  final-shift-cipher.py  encryption.py  file.txt
31 def decrypt_ciphertext(ciphertext, dictionary):
32     message_space = "abcdefghijklmnopqrstuvwxyz "
33     char_values = {char: i + 1 for i, char in enumerate(message_space)}
34
35     frequencies = Counter(ciphertext.lower()) # Convert ciphertext to lowercase for case-insensitive comparison
36     sorted_frequencies = sorted(frequencies.items(), key=lambda x: x[1], reverse=True)
37
38     most_frequent_char = sorted_frequencies[0][0]
39     shift_e = char_values[most_frequent_char] - char_values['e']
40     shift_space = char_values[most_frequent_char] - char_values[' ']
41     shift_a = char_values[most_frequent_char] - char_values['a']
42     shift_r = char_values[most_frequent_char] - char_values['r']
43     shift_i = char_values[most_frequent_char] - char_values['i']
44     shift_o = char_values[most_frequent_char] - char_values['o']
45     shift_t = char_values[most_frequent_char] - char_values['t']
46     shift_n = char_values[most_frequent_char] - char_values['n']
47     shift_s = char_values[most_frequent_char] - char_values['s']
48     shift_l = char_values[most_frequent_char] - char_values['l']
49     shift_c = char_values[most_frequent_char] - char_values['c']
50     shift_u = char_values[most_frequent_char] - char_values['u']
51
52     decrypted_text_e = ""
53     decrypted_text_space = ""
54     decrypted_text_a = ""
55     decrypted_text_r = ""
56     decrypted_text_i = ""
57     decrypted_text_o = ""
58     decrypted_text_t = ""
59     decrypted_text_n = ""
```



```

for char in ciphertext:
    if char.lower() in message_space: # Check lowercase character against message space
        new_value_e = char_values[char.lower()] - shift_e
        new_value_e = (new_value_e - 1) % len(message_space)
        decrypted_text_e += message_space[new_value_e]

        new_value_space = char_values[char.lower()] - shift_space
        new_value_space = (new_value_space - 1) % len(message_space)
        decrypted_text_space += message_space[new_value_space]

        new_value_a = char_values[char.lower()] - shift_a
        new_value_a = (new_value_a - 1) % len(message_space)
        decrypted_text_a += message_space[new_value_a]

        new_value_r = char_values[char.lower()] - shift_r
        new_value_r = (new_value_r - 1) % len(message_space)
        decrypted_text_r += message_space[new_value_r]

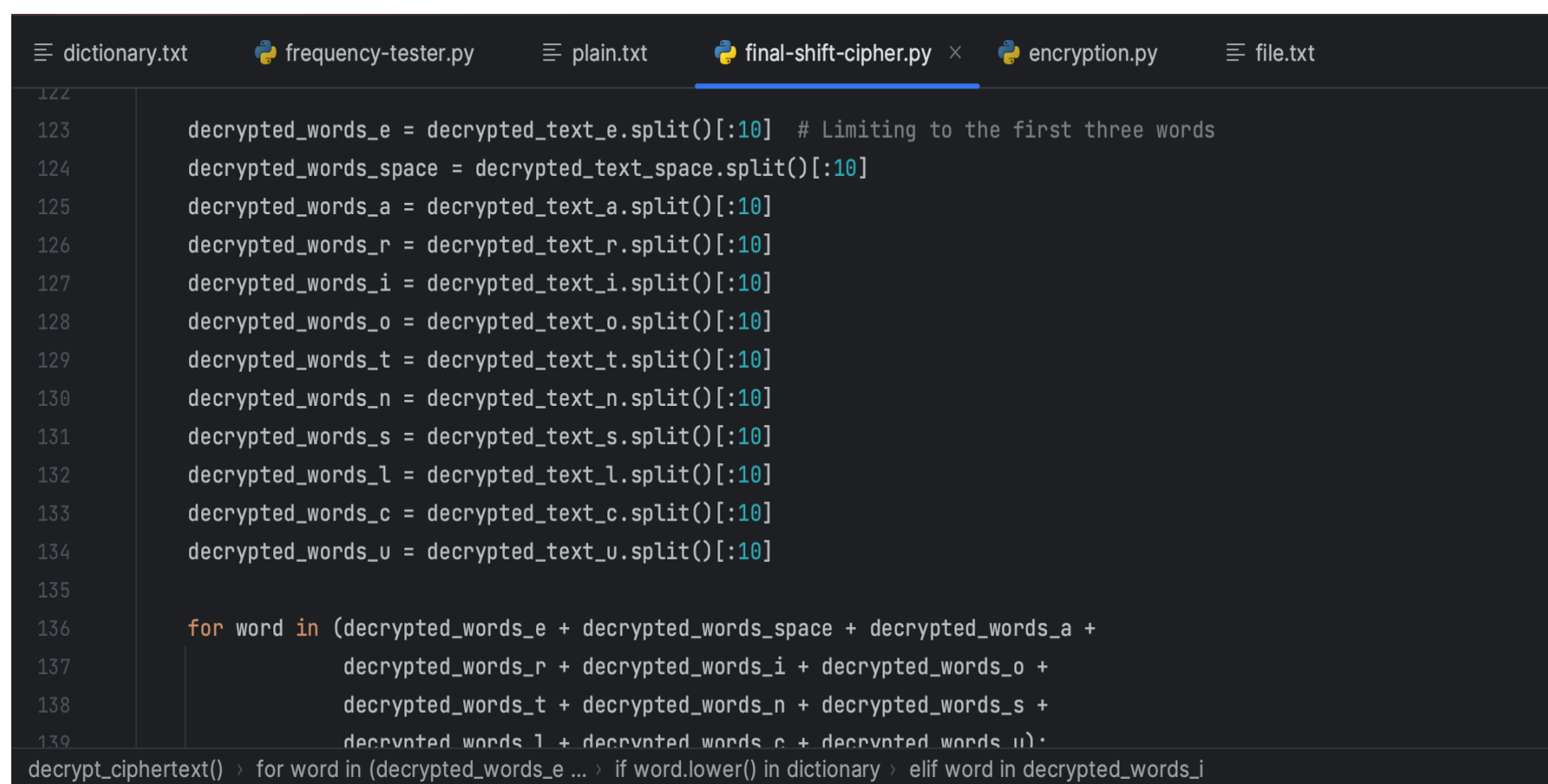
        new_value_i = char_values[char.lower()] - shift_i
        new_value_i = (new_value_i - 1) % len(message_space)
        decrypted_text_i += message_space[new_value_i]

        new_value_o = char_values[char.lower()] - shift_o
        new_value_o = (new_value_o - 1) % len(message_space)
        decrypted_text_o += message_space[new_value_o]

        new_value_t = char_values[char.lower()] - shift_t
        new_value_t = (new_value_t - 1) % len(message_space)
        decrypted_text_t += message_space[new_value_t]

```

4.4 Validation:



```

122
123     decrypted_words_e = decrypted_text_e.split()[:10] # Limiting to the first three words
124     decrypted_words_space = decrypted_text_space.split()[:10]
125     decrypted_words_a = decrypted_text_a.split()[:10]
126     decrypted_words_r = decrypted_text_r.split()[:10]
127     decrypted_words_i = decrypted_text_i.split()[:10]
128     decrypted_words_o = decrypted_text_o.split()[:10]
129     decrypted_words_t = decrypted_text_t.split()[:10]
130     decrypted_words_n = decrypted_text_n.split()[:10]
131     decrypted_words_s = decrypted_text_s.split()[:10]
132     decrypted_words_l = decrypted_text_l.split()[:10]
133     decrypted_words_c = decrypted_text_c.split()[:10]
134     decrypted_words_u = decrypted_text_u.split()[:10]
135
136     for word in (decrypted_words_e + decrypted_words_space + decrypted_words_a +
137                 decrypted_words_r + decrypted_words_i + decrypted_words_o +
138                 decrypted_words_t + decrypted_words_n + decrypted_words_s +
139                 decrypted_words_l + decrypted_words_c + decrypted_words_u):
decrypt_ciphertext() > for word in (decrypted_words_e ... > if word.lower() in dictionary > elif word in decrypted_words_i

```

Importance: Validation serves as the final checkpoint in our decryption process, ensuring the integrity and accuracy of the deciphered output. The provided code snippet exemplifies this validation procedure by checking the first three words of the decrypted text against a loaded dictionary of English words. If any of these words are found within the dictionary, it serves as confirmation of the validity of the decryption, and the decrypted text is returned as the final output. This crucial step instills confidence in the decrypted message, affirming its coherence and linguistic authenticity.

4.5 Analysis and Discussion:

1. Loading the Dictionary:

- Open the dictionary file.
- Read each word and convert it to lowercase.
- Store these words in a set for efficient lookup during decryption.

2. Decrypting the Ciphertext:

- Define the message space, which includes lowercase letters and the space character.
- Assign numerical values to each character in the message space to facilitate shifting.
- Analyze the frequencies of characters in the ciphertext to identify the most frequent character.
- Calculate shift values based on the most frequent character to potentially decrypt the ciphertext.
- Perform decryption using the calculated shift values.
- Validate the decrypted output by checking if any words match the words in the loaded dictionary.
- If a valid word is found, consider that decryption as a potential plaintext.
- If no valid words are found among the potential decryptions, default to the first decryption.

3. Main Function:

- Load the dictionary file containing valid English words.
- Prompt the user to input the ciphertext to be decrypted.
- Utilize the decryption algorithm to decrypt the ciphertext and attempt to match it with words from the dictionary.
- Print the decrypted plaintext if a valid word is found or print the default decryption otherwise.

This approach breaks down the decryption process into steps such as loading the dictionary, analyzing the ciphertext, decrypting using calculated shifts, validating the decrypted output, and handling the main decryption process.

Conclusion:

The effectiveness of the presented algorithms hinges on the assumption of linearity in the key schedule, particularly in the j function. This assumption is pivotal for ensuring that subsequences determined by the key length are offset uniformly, leading to near-instant execution times for the algorithm. However, this assumption may not always hold true, especially in the case of non-linear key scheduling algorithms. To address this limitation, one potential avenue for improvement involves enhancing the `find_key_length` method by exploring different key scheduling algorithms, such as iterating through polynomials of varying degrees. By modeling the key scheduling algorithm, it may be possible to determine an inverse key with respect to this model, thereby improving the algorithm's adaptability to different encryption schemes.

Another area ripe for improvement lies in the correctness of decryption. As the second algorithm relies on statistical methods, there exists a possibility that the decrypted output may appear as garbage text if it is incorrect. This underscores the need for enhancing the algorithm's accuracy, particularly in cases where the input dictionary is not utilized. To mitigate this issue, the algorithm could maintain a list of candidate keys and key lengths, systematically trying each one until a valid word from the dictionary is obtained. Although this approach may incur higher computational costs, it offers the promise of significantly enhancing the accuracy and reliability of the decryption process.

References:

- [1] Neuenschwander, D. (2004). Probabilistic and Statistical Methods in Cryptology: An Introduction by Selected Topics (Vol. 3028). Springer Science & Business Media.
- [2] Friedman, W.F. (1922). The Index of Coincidence and Its Applications in Cryptology.
- [3] Phamdo, N. (n.d.). Statistical Distributions of English Text. Retrieved from <https://web.archive.org/web/20171212040428/http://www.data-compression.com/english.html>
- [4] <https://www3.nd.edu/~busiforc/handouts/cryptography/letterfrequencies.html>